

Versioning System for Distributed Ontology Development

Suresh K. Damodaran

February 02, 2016

Distribution A: Public Release

EXECUTIVE SUMMARY

Common Cyber Environment Representation (CCER) is an ontology for describing operationally relevant, and technically representative, cyber range event environments. Third-party ontology developers as well as in-house ontology developers contributed to CCER Ontology. Since the cyber range environments continue to evolve and expand, we expect the CCER Ontology also to evolve and expand. Therefore, an easy to comprehend versioning scheme that will support a systematic ontology evolution is needed. This document describes the requirements for such a versioning scheme, and its design. This document also describes how to assign version numbers under different ontology evolution situations, and provides guidelines for evaluating the impact of the version changes.

ACKNOWLEDGMENTS

The author would like to express his gratitude to Jeffrey M. Bradshaw, Larry Bunch, and Andrzej Uszok of Florida Institute for Human and Machine Cognition (IHMC) for substantial contributions to the development of the ontology versioning scheme described in this document.

This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.

© 2016 Massachusetts Institute of Technology.

Delivered to the US Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

TABLE OF CONTENTS

	Page
EXECUTIVE SUMMARY	III
ACKNOWLEDGMENTS	V
LIST OF FIGURES	IX
1. INTRODUCTION	1
2. CCER ONTOLOGY ORGANIZATION	7
2.1 Ontology Taxonomy	7
2.1.1 Origination Based Taxonomy	7
2.1.2 Usage Based Taxonomy	8
2.2 Ontology Modularity	11
2.2.1 Bridging	11
2.3 File System Alignment	12
2.3.1 CCER Folder Structure	12
2.3.2 Namespaces	14
3. VERSIONING SCHEME	15
3.1 Topic Evolution and Folder Structure	15
3.2 Encoding Ontology Versioning	17
3.2.1 Deprecation Information	17
3.3 No Delete Policy	19
4. ONTOLOGY EVOLUTION	21
4.1 Version Assignment By Developers	21
4.2 Ontology Evolution for Users	24
5. CONCLUSION	26

LIST OF FIGURES

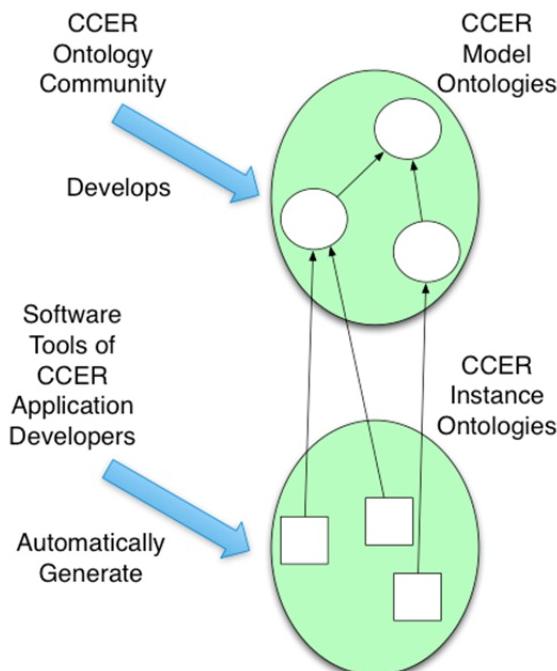
Figure No.	Page
-----------------------	-------------

No table of figures entries found.

1. INTRODUCTION

Within the cyber range community, ontologies are being used increasingly to specify the description of the environment in which cyber events can be conducted [CSTL1]. This paper describes some key design considerations we have formulated for the evolution of one set of such ontologies for that community, the Common Cyber Environment Representation (CCER) [CCER1]. Common Cyber Environment Representation (CCER) is an ontology for describing operationally relevant, and technically representative cyber range event environments. CCER Ontology is created to support reusable machine-interpretable representation of the cyber range event environment that will enable automation of the event process through interoperability among cyber range support tools [CRIS1].

Note on terminology: We use the term *CCER Ontology* or sometimes the abbreviated form *CCER* to refer to all ontology files within CCER. The term *ontologies* refers to multiple ontology files, while, the term *ontology* refers to a single ontology file.



CCER Ontology consists of a set of ontologies in RDF/OWL [RDF1, OWL3] to represent the cyber event environment. A subset of CCER Ontology are considered *model* ontologies, or *CCER OWL schema*, and the rest of the ontologies are considered *instance* ontologies. The model ontologies define the framework for describing the various aspects of a cyber range event environment. The instance ontologies describe individual cyber event environments using the framework provided by the model. Cyber range developers use software tools to create a specifically configured instance of the model that other tools may use to do further processing. This instance is referred to as an *instance ontology* to distinguish it from the CCER *model ontologies* to which the instance ontology conforms. The scope of this document is limited to the versioning of CCER model ontologies.

Event environments continually evolve to include newer kinds of entities, systems, and simulations and the CCER Ontology must be modified to incorporate these new developments. The CCER Ontology, much like source code, will also change for other reasons, including the following:

- bugs
- changes in the

Figure 1: Model and Instance Ontologies

- domain such as changes to organizations, changes in names of things in the world
- changes in a community's shared conceptualization of the domain that results in splitting and merging of individual classes or properties; or refactoring of groups of classes and properties
- changes in the way the ontology will be used, for example, different perspectives on the domain such as natural vs. man-made features on a map; or continuous vs. discrete time
- restructuring of the ontology motivated by performance considerations for a given application or due to the requirements of a given reasoner.

While cyber range ontology developers share many requirements in common with other developers of large ontologies in loosely coupled development communities, they also deal with considerations specific to the defense cyber range community, since cyber range ontology developers and users belong to that community. Below is a list of the major requirements that govern the evolution of CCER Ontology:

- *Interoperability across versions.* Tools using CCER may evolve to use several different versions of subsets of CCER Ontology. In a heterogeneous environment of tools, it is normal to expect

different tools or toolsets to interoperate even though they use different versions of a subset of CCER. It is important to have the ability to clearly identify which versions of an ontology or a set of ontologies have backward compatibility, and which do not.

- *Distributed and isolated development.* CCER users and developers are diverse and dispersed across multiple organizations. Projects that use CCER have different sponsor organizations. Security requirements considerably limit information and result sharing across projects. Often, a team may independently modify the same concepts or relationships without the knowledge of anyone else in the community. Such modifications may be due to specialized subject matter expertise or merely because of an incompatibility between some portion of the ontologies and the software tools in use at a given location. These modifications may or may not be shared with other CCER ontology developers. For these and other reasons we are entirely dependent on community conformance to a clear set of development and versioning guidelines to assure that changes and extensions can be integrated back into the “main development branch” when and if they become available.
- *Long accreditation process.* New tools and technologies go through a long accreditation process in Department of Defense environments. Pinpointing the changes from one set of CCER Ontology to a new set of CCER Ontology will accelerate the accreditation process when a tool upgrades to a newer version of the CCER Ontology.
- *Reuse of third-party ontologies.* To expedite the development of CCER Ontology, the CCER team has reused a number of publicly available ontologies. These ontologies have separate independent evolutionary modification and versioning cycles and conventions. When CCER Ontology are dependent on these external ontologies, there must be a way to make CCER Ontology adapt to the version changes of external ontologies gracefully.
- *Version management complexities.* Versioning, especially sophisticated versioning schemes, can be problem prone because the complicated rules confuse developers and users. Some of the problems are the following: version-bloat, where copies of older versions litter the landscape; version-fragmentation, where one must jump from folder to folder to follow the fragmented history of changes to given file; and version amnesia, where nobody can remember what version number to use for a given ontology and has to look it up constantly. Use of simple rules and automation tools can considerably help with version management.
- *Use case ontology configuration.* Each cyber event environment may rely on a different subset of the CCER Ontology. The use case captures the exact ontology configuration required for a particular type of project, for example, Situational Awareness. The ontology configuration will include the model ontologies, their versions, and their import dependencies. The details of the

configuration are used by software tools to verify conformance of an instance ontology to the corresponding model ontologies. In addition, CCER Ontology users may be able to use the configuration information to learn how the instance ontology and corresponding model ontologies evolved over a period of time.

Many of the requirements listed above may be relevant to many members of the Semantic Web community. For example, the *distributed and isolated development* requirement may apply to non-cyber range communities of public ontology users and developers working separately who may rarely or ever be in contact with one another. Their separation by time and distance mirrors the separation by sponsors and security classifications experienced by the cyber range community. Obviously, the long accreditation time for secured environments is unique for the cyber range community. The requirement related to reuse is the consequence of a development choice that CCER team adopted.

In the cyber range community, the most practical means to collaborate on ontologies across teams is by sharing files asynchronously, outside of a version management system such as CVS, git, or svn. Collaborative development of ontologies assume real-time collaboration using tools [Noy1, LHK09, RGZH, GJKZ, RGHB]. The prevalent ontology development approaches assume an open environment for sharing of information across the Internet, or the “semantic web.” However, we observe that the maintenance of an ontology and its reuse is not a high priority for the majority of the publicly available ontologies. For example, Swoogle [SEA1], the first web search tool for ontologies, lists several ontologies created for specific domain areas, and it would be hard for the user to know which one is maintained or widely adopted, and how to use the available ontology without researching each ontology. There could be multiple versions for an ontology, but the onus of making sure that the right version is used for a specific application is on the user of the ontology. A clear set of guidelines for evolution of an ontology would have considerably helped the users of the ontology in these situations. The currently accessible guidelines such as [VURI] do not provide clear and adequate guidance for versioning ontologies developed spread out in space and time.

The OWL 2 representation provides new features specifically to address requests for versioning support [OWL2]. However, it is important for our ontology consumers to always specify the desired major version of every ontology import statement by including the major version number in the ontology IRI as well as the location (URL). Several approaches to ontology version management require specific tools to be used for version change detection [OVR1]. While CCER certainly encourages the use of such tools, such a requirement could be raised as a barrier to entry for the community. An approach is needed that does not require new tools.

To address the requirements above, we have defined a set of guidelines for version evolution of CCER ontology for its maintenance and extension. We also hope that this set of guidelines may be applicable to other ontologies that aspire to be better maintained under similar requirements.

2. CCER ONTOLOGY ORGANIZATION

In this section, we describe the organization of the CCER Ontology. The organization is described in three subsections. First, we describe how we categorize the different components of CCER Ontology in Section 2.1. In Section 2.2, we describe the modular structure for CCER Ontology based on the concept of Topic. Section 2.3 defines the close alignment between the file system structure and the CCER Ontology organization described in Sections 2.1 and 2.1.

2.1 ONTOLOGY TAXONOMY

In this section we describe the taxonomy of CCER Ontology. CCER Ontology contents can be classified based on two useful criteria: origination and usage. There are two broad categories based on the origin of the ontologies: ontologies created by the CCER team, or third-party ontologies that are not created by CCER team. The CCER Ontology can be categorized as Core, Domain, and Use Case, based on their usage. We first describe in Section 2.1.1, how CCER Ontology is categorized based on where its components were developed. In Section 2.1.2, we describe another orthogonal categorization based on how ontologies are used. Both of these classifications are used in Section 2.2 and 2.3 to organize and store CCER Ontology.

2.1.1 Origination Based Taxonomy

Several useful ontologies have been created over the last decade, and CCER utilizes those ontologies. Using or extending classes and properties from an existing third-party core ontology is always preferred over inventing new ontologies that would practically serve the same purpose. Reusing existing ontologies does not simply save time and effort to CCER team, it also increases the prospects of interoperability of CCER Ontology with other ontologies. The third-party ontologies exhibit varied properties, and how CCER team maintains these ontologies depends on their respective properties.

1. Some of these ontologies are actively being maintained, such as KAoS [UBJ2004] or OWLTime [TIME]. CCER will redistribute those ontologies without any change as long as the copyright statements permit such re-distribution. When such redistribution is not possible by CCER, the distribution will contain pointers as to where to obtain those ontologies.
2. When an ontology is not actively maintained, and CCER team will need to modify those, CCER team will do such updates to those ontologies only if copyright statements permit it.
3. When an ontology is obtained from other formats or transcribed from papers, CCER will consider those ontologies to be CCER team created or third-party generated, depending on

whether CCER team is responsible for generation and maintenance of the ontology. For example, NML [NML] is maintained externally, and therefore, we consider it to be a third-party ontology.

When a particular ontology has a third-party origin, and already has an assigned namespace URI, CCER will continue to use the third-party ontology namespace URI. The CCER Ontology team will make reasonable efforts to maintain or update CCER Ontology consistent with new versions of ontologies developed by standards groups or other third parties as they appear. CCER team will avoid directly modifying third-party ontologies unless CCER team is directly responsible for their maintenance.

2.1.2 Usage Based Taxonomy

Based on the usage of the ontologies, CCER Ontology can be classified as Core, Domain, or Usecase. We describe these categories below. We also show how the origination based taxonomy and usage based taxonomy relate to each other.

Core ontologies define general-purpose classes that are used by other, more specific Domain ontologies. Domain ontologies define domain-specific classes and properties that are specific to an area of knowledge or user application (e.g., ontologies defining firewalls, networks, flowers). Use case ontologies contain instances of Domain and Core ontologies, and do not define any new classes or properties. Use case ontologies are created to support specific use cases.

Core Ontologies

A core ontology defines general-purpose classes that are widely-used by other, more specific Domain ontologies. Upper ontologies are considered to be core ontologies. Because changes to core ontologies can have significant side effects on other CCER ontologies that depend on them, they are not updated too frequently.

Third-Party Core Ontologies. Third-Party core ontologies are specified and maintained by standards groups or other third parties (e.g., OWL Time, Dublin Core, KaOS core ontologies, and WSGS 84).

Domain Ontologies

Domain ontologies define domain-specific classes and properties that are specific to an area of knowledge or user application (e.g., ontologies defining firewalls, networks, flowers). Domain ontologies may be created by specialization of classes in core ontologies by extending them with new subclasses and properties. Domain Ontologies can be used as the main building blocks of a given CCER application (e.g., ontologies defining firewalls or networks).

Domain Ontologies may frequently have dependencies among themselves. In such cases cycles need to be avoided among such dependencies. In addition, no transitive dependencies should occur among ontologies: i.e., if one ontology imports another, it should not directly or indirectly import an ontology on which the reused one is dependent. Sometimes several domain ontologies will share a dependency on classes in a Core ontology or other Domain ontologies.

Other than code lists, such as a list of all fifty states; or singleton values, such as *true* or *false*, Named Individuals should not normally be included in domain ontologies. Such Named Individuals may also be defined as part of core and upper ontologies. Domain Ontologies sometimes may have significant side effects on other CCER Ontology that depend on them, so they should be modified thoughtfully and with care.

CCER Domain Ontologies. This category is meant for ontologies that are specified and maintained by the CCER Ontology Team (e.g., CCER Firewall Ontologies). This may also include ontologies that have been specified elsewhere but are actively maintained and updated by CCER (in some cases because they are not being actively maintained by their creators; or in the other case where the original format is not OWL (e.g., SCAP¹)).

Third-Party Domain Ontologies. This category contains ontologies that are specified and maintained by standards groups or third parties (e.g., NML Network Ontologies).

Use Case Ontologies

Use case ontologies contain instances of Domain and Core ontologies, and do not define any new classes or properties, and only contain instance data such as NamedIndividuals and their properties. These ontologies serve two primary purposes. First, to define instance ontologies to support a use case. The use case instances are used as examples so tools implementing that use case may generate similar use case instances through automation. Ontology instance management tools can also be defined based on the use case instance ontologies (e.g., diffs between instances of different versions). Second, the use case instances define the configuration (classes, properties, and individuals, properties) of all ontologies imported to support a use case. This configuration will help to develop tools for managing the ontologies (e.g., to support a version upgrade).

¹ SCAP is maintained as an ontology by CCER though it was created in XML form by external parties.

2.2 ONTOLOGY MODULARITY

CCER Ontology is designed to be modular. The unit of modularity in CCER is a *Topic*. A Topic contains multiple ontology files that are closely related. This close relationship implies that when one of the ontology files changes within a Topic, it is likely that many other files in that Topic also need to be changed. When third-party ontologies are imported, the entirety of the ontology is considered a Topic in CCER. However, what constitutes a Topic when ontologies are designed by CCER team is very much dependent on the subject of the Topic.

New Topics can be added to CCER Ontology with relative ease. Owners, who are responsible for its upkeep, can be assigned to specific Topics. To support distributed development of the Topics the remote and off-line modifications to CCER Ontology must be eventually integrated with CCER Ontology.

A Topic may contain another Topic. Such inclusion occurs when a specific Topic grows in complexity, and it makes more sense to split one Topic into multiple Topics. For example, a *network* Topic may evolve into *layer3_network* and *layer2_network*. When a Topic is split into multiple Topics, considerations such as the impact of the new namespaces need to be addressed.

When an ontology needs to be extended, there may be a need to not to change the original ontology files. This kind of extension may occur in multiple circumstances, as enumerated below.

1. When a third-party ontology that is maintained external to CCER is extended,
2. When a CCER originated ontology is widely used by many other ontologies, and a change directly to this ontology may trigger an undesirable broader update to CCER Ontology, or
3. When two unchangeable ontologies need to be integrated.

We use a technique called *bridging* in these circumstances to extend ontologies. We discuss bridging next.

2.2.1 Bridging

A bridge is essentially an ontology created by aligning multiple ontologies. When third-party ontologies cannot be modified due to IP restrictions or because the ontologies are maintained actively, a bridge ontology containing classes that connect the (unchangeable) upper ontology to the (unchangeable) third-party ontology or ontologies should be defined. The name of the bridge ontology must reflect all of the ontologies that it is directly importing for the purpose of bridging. As an example, the bridge ontology between KaOS (core ontology) and NML (domain ontology) defines subclass relations between the NML classes to the KaOS classes. In this bridge ontology, the NML class of *NetworkObject* is defined as a subclass of more generic KaOS *ComputingEntity* class.

Where necessary, bridge ontologies can be defined in order to relate third-party ontologies to other CCER Ontology. Let us consider two ontologies, a “first ontology,” and a “second ontology.” In the bridge ontology, the following definitions can be placed to bridge the first ontology to the second ontology:

- Define equivalency of a class from the first ontology to a class in the second ontology,
- Define a class from the first ontology as a subclass of a class in the second ontology,
- Define the range of a property defined in the first ontology using a class in the second ontology,
- Adding a new property to a class in first ontology with its domain or range defined using the class from the second ontology.

The bridge ontology should not define new classes or create properties which have domain and range in the same ontology; either first or second. The intent of the bridge ontology is not to define new classes but to extend definitions of existing classes in the linked ontologies but using the classes from the other ontology. For example the bridge between the host and network ontologies defines the new property of the host such as *hasIPAddress*, *hasMACAddress*, *hasSubnet* which have ranges defined using the classes from the network ontologies.

2.3 FILE SYSTEM ALIGNMENT

CCER Ontology is organized using the concept of Topics and ontology files within those Topics. Corresponding to each Topic, there is exactly one folder in CCER Ontology. The namespace assigned to ontologies created by CCER team has direct correspondence to the folders and the ontology file name. We describe the folder structure first, followed by name spaces.

2.3.1 CCER Folder Structure

The topmost folder, *ccerschema*, for CCER Ontology will contain three subfolders, *core*, *domain*, and *usecase*, corresponding to Core, Domain, and Usecase ontologies.

The CCER Core ontologies will be in folder *ccerschema/core/ccer*. The ontologies that go into this subfolder are maintained by CCER ontology team members. Third-Party core ontologies that are specified and maintained by standards groups or third parties (e.g., OWL Time, Dublin Core, KaOS core ontologies, and WSGS 84) are in folder *ccerschema/core/third_party*. A third-party ontology will be

considered a single Topic, and will be assigned a Topic and a folder name corresponding to the third-party. For example, in Figure 2, see the folder *foaf*² corresponding to Friend-Of-A-Friend ontology.

The Domain ontologies will be in folder “ccerschema/domain” Domain ontologies created by CCER team are in folder *ccerschema/domain/ccer*. Third-party Domain ontologies will be in folder *ccerschema/domain/third_party*. Usecase ontologies are in folder *ccerschema/usecase*. Third-party Usecase ontologies will be in folder *ccerschema/usecase/third_party*. Topics maintained by CCER team are in the *ccer* subfolders of core, domain, and use case folders. Topics that are not maintained by CCER are in third-party subfolders of *core*, *domain*, and *usecase* folders. The following diagram illustrates the folder structure for the CCER Ontology.

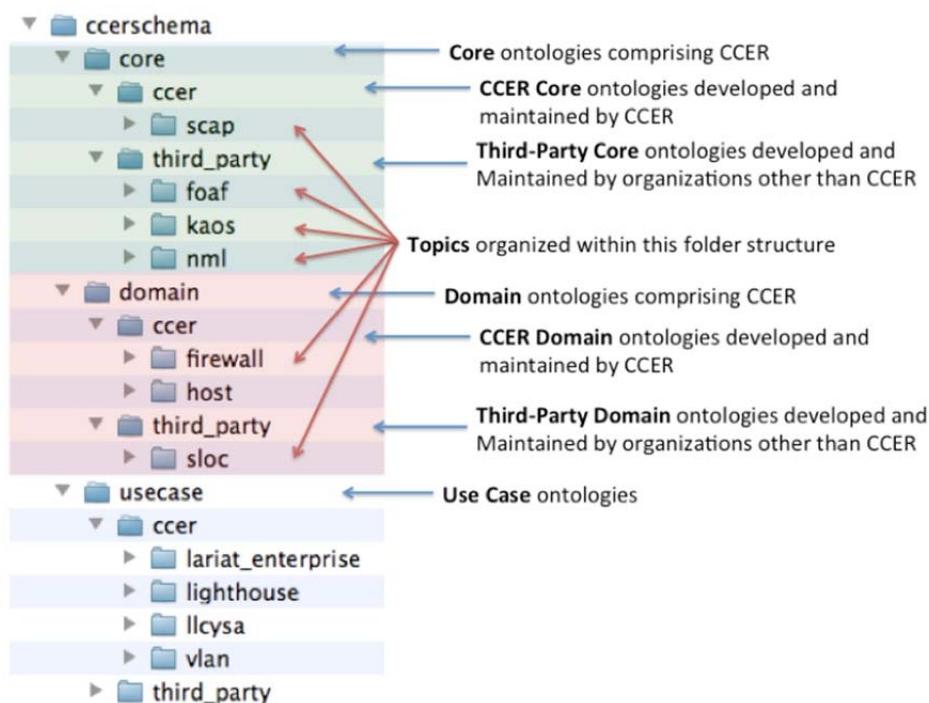


Figure 2: CCER Folder Structure

The top-level folders (*core*, *domain*, and *usecase*) are defined by CCER to classify topics according to content. Within the *core* and *domain* folders the second level further classifies Topics into those developed by CCER (e.g. *event*, *scap*, *firewall*, *host*, etc.) and Topics created by a third party and reused

² <http://www.foaf-project.org/>

by CCER (e.g. *nml*, *ndl*, *foaf*, *KaOS*, etc.). Each use case is considered a Topic and will be defined in its own folder underneath the *usecase* folder.

2.3.2 Namespaces

The namespace of an ontology in CCER is defined in its URI. For CCER maintained ontologies URIs³ will follow this format:

```
http://ll.mit.edu/<YEAR>/ccer/<ontology_type>/<topic_name>[/<topic_name>]<major_version_number>/<ontology_file_name>
```

Ontologies within a Topic also will share the same namespace URI prefix. Here, <YEAR> is the year of creation of the ontology, <ontology_type> is one or core, domain, or usecase, <topic_name> is the name of the Topic, [/<topic_name>] is a series of Topic names separated by “/,” <major_version_number> will have the form *v*<integer>, starting with *v1*. The <ontology_file_name> is the actual name of the file in which an ontology exists. At least one *topic_name* is necessary, and multiple <topic_name>s occur when there are nested Topics. Below is the pathname to the ontology file, and note that the namespace scheme maintains a direct correspondence to the file structure.

```
ccerschema/<ontology_type>/<topic_name>[/<topic_name>]<major_version_number>/<ontology_file_name>
```

As stated earlier, third-party ontologies will have the third-party assigned URI, and CCER will not alter those URIs.

³ IRI is a generalized version of URI that supports international characters, and it may be used in place of URI if the tool set supports it.

3. VERSIONING SCHEME

Every ontology file will be assigned a full version number consisting of three numbers separated by decimal points (e.g. v1.2.0) where the leftmost number is the *major version*, the middle number is the *minor version*, and rightmost number is the *pico version*. By convention, the major version starts with v1 while the minor and pico versions start with 0. Therefore the first release of a new ontology should be v1.0.0 to indicate the first major release with no incremental revisions.

As a matter of practice, minor version numbers are used for modifications thought to be generally backward-compatible or limited in impact to applications and other ontologies. Major numbers are used for incompatible versions or changes that may have wider impact. The pico number is used for internal CCER development versioning only, and is therefore not exposed to end users and is not indicative of whether the changes are backward compatible.

Only the major version number is used in the URIs for the ontologies. The full three-digit version number is always included as an annotation within the content of the ontology file as explained in the next section. The major version in the URI should always match the major version annotated within the file. Indicating the full version in the contents rather than the URI allows for tracking incremental changes to the ontology files using the minor and pico version numbers without changing the URI, and thus the identity, of the ontology classes.

3.1 TOPIC EVOLUTION AND FOLDER STRUCTURE

In CCER Ontology, a Topic is the biggest unit that is versioned together. The release version of CCER Ontology as a whole may not be defined, or, if defined, may have no correspondence to the versions of the Topics within it. The ontologies in a Topic may be versioned as v1, v2, etc., and the folders corresponding to each version is called a “Version folder.” Every Topic contains one or more version folders labeled with the major version of that Topic. Below are the Topic versioning rules.

1. All of the ontologies within a version folder in a Topic share the same major version number.
2. When some of the ontologies in a Topic need to be updated with a new major version, a new version folder is created, and only the updated ontologies are stored in the new version folder. The contents of the folder in the original Topic are untouched. In contrast, minor and pico version changes are performed in place by incrementing the version value within the ontology file each time it is modified.

- When a new Topic is created with a subset of ontologies of the original Topic, a new Topic folder is created within the original Topic folder, and the selected subset of ontologies are added in a version folder, “v1,” inside the new Topic folder. The contents of the folder in the original Topic is untouched.

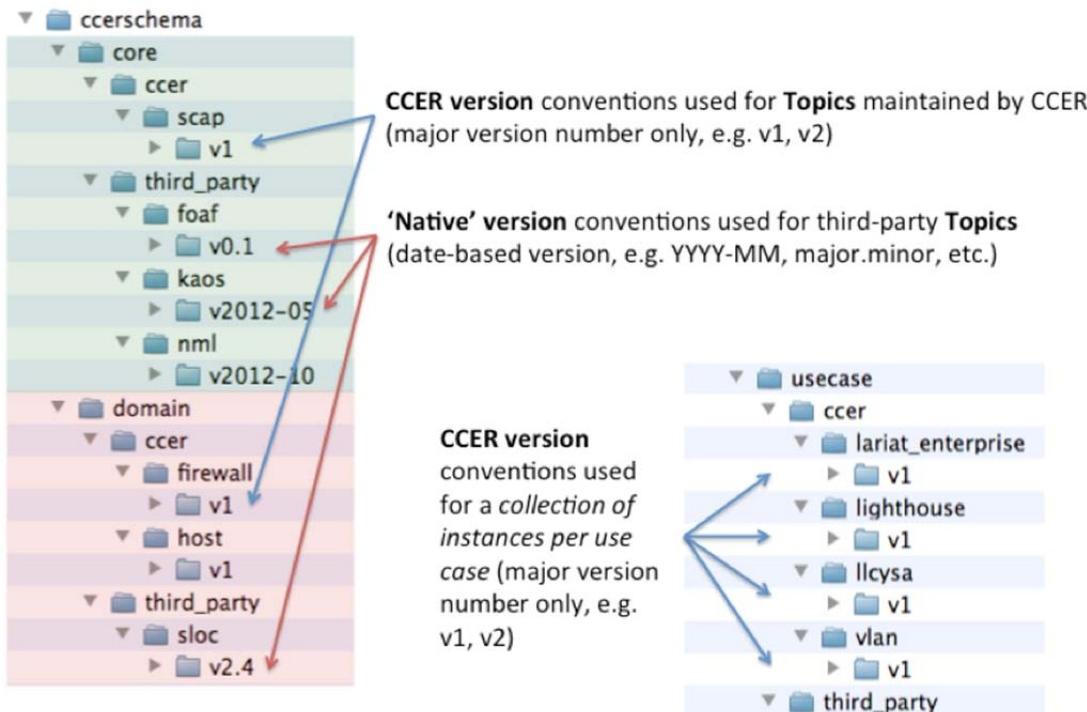


Figure 3: Major Versions

One key advantage of independently versioning each Topic is that multiple major versions of the Topics in CCER can coexist in every release. The version subfolders will always start with the letter ‘v’ to distinguish version-labeled folders from other folders. For Topics developed by CCER team, the ‘v’ is followed by an integer signifying the major version of that Topic (e.g. v1, v2, v3). For third party Topics maintained by CCER, the version folder names still start with ‘v’, but otherwise adopt the ‘native’ version scheme used by the third party.

Third-party Topic version conventions may vary but are typically based upon some numbering scheme (e.g. v1, v1.2, v1.2.0, v3.0.1b2) or the release date (e.g. vYYYY-MM as in v2012-05). For example, in Figure 3, CCER version conventions (i.e., “v1”) are used for the *host*, and *firewall* ontologies. On the

other hand *foaf* is shown with its native version numbering scheme of *v0.1* and *KaOS* is shown with a version number of *v2012-06*.

The *usecase* folder contains collections of example instances that serve as guidelines for how the CCER ontology should be used in various contexts (e.g. *lariat*, *lighthouse*, *llcysa*, and *vlan*). Each of these use case folders contains one or more version subfolders starting with the letter ‘v’ followed by an integer signifying the major version of that use case. No new classes or properties should be defined by any of the ontology files in the *usecase* folder.

CCER versioning scheme used for the ontologies in *core* and *domain* may be used for ontologies in *usecase* folder, though we do recognize that some non-CCER users may develop their own versioning schemes.

3.2 ENCODING ONTOLOGY VERSIONING

In this section, we describe how version information is encoded or embedded in the ontology files under multiple situations.

Version Embedding

Version numbers and their properties are embedded in the ontology files. Below are examples of how such embedding is done.

Version Number

Version number (major.minor.pico) are embedded in every ontology file as *owl:versionInfo* property⁴. The following example illustrates using *owl:versionInfo* which is included for the *owl:Ontology* node only:

```
<owl:Ontology rdf:about="http://sample.org/sample.owl">
  <owl:versionInfo>1.0.0</owl:versionInfo>
</owl:Ontology>
```

3.2.1 Deprecation Information

To indicate, that a class, property, or instance is deprecated during a minor version upgrade, the *owl:deprecated* Annotation Property is used. This Annotation Property is not used if there are no classes, properties, or instances that are not deprecated in an ontology file. Some ontology constructs cannot

⁴ <https://www.w3.org/TR/owl-ref/#versionInfo-def>

be deprecated in this way including cardinality, range, and transitivity because there are no URIs to deprecate. Additional custom Annotation Properties are defined specifically for CCER developers to indicate the reason for depreciation as well as the provenance of the depreciation (e.g., author).

The following example OWL syntax defines the additional depreciation annotation properties that developers can use to document the reason and author of the depreciation for any class, property, or individual.

```
<owl:AnnotationProperty rdf:about="http://sample.org/sample.owl#deprecationReason">
  <rdfs:label xml:lang="en">deprecation reason</rdfs:label>
</owl:AnnotationProperty>

<owl:AnnotationProperty rdf:about="http://sample.org/sample.owl#deprecationAuthorEmail">
  <rdfs:label xml:lang="en">deprecation author email</rdfs:label>
</owl:AnnotationProperty>
```

The next example illustrates deprecating an OWL class using *owl:deprecated* as well as providing the custom annotations indicating the reason for depreciation and the user.

```
<owl:Class rdf:about="http://sample.org/sample.owl#SampleClass">
  <owl:deprecated rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</owl:deprecated>
  <deprecationReason rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Replaced by
http://sample.org#new\_sample.owl#NewSampleClass</deprecationReason>
  <deprecationAuthorEmail
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">jsmith@sample.com</deprecationAuthorEmail>
</owl:Class>
```

The next example similarly illustrates the depreciation of a property.

```
<owl:ObjectProperty rdf:about="http://sample.org/sample.owl#sampleProperty">
  <owl:deprecated rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</owl:deprecated>
  <deprecationReason rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Replaced by
http://sample.org#new\_sample.owl#newSampleProperty</deprecationReason>
  <deprecationAuthorEmail
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">jsmith@sample.com</deprecationAuthorEmail>
</owl:ObjectProperty>
```

This final example shows the depreciation of a Named Individual.

```
<owl:NamedIndividual rdf:about="http://sample.org/sample.owl#sampleIndividual">
  <owl:deprecated rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</owl:deprecated>
  <deprecationReason rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Replaced by
http://sample.org#new\_sample.owl#newSampleProperty</deprecationReason>
  <deprecationAuthorEmail
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">jsmith@sample.com</deprecationAuthorEmail>
</owl:NamedIndividual>
```

3.3 NO DELETE POLICY

A distribution of CCER Ontology will consist of the current and earlier versions of an ontology. This policy allows a guarantee of complete continuity for members of the community who, for one reason or another, must continue to use older versions.

4. ONTOLOGY EVOLUTION

The evolution of CCER ontology needs attention from both ontology developers and its users. Ontology developers will need to assign version numbers appropriately as ontologies evolve, and the users will need to understand how a version change effects their activities and tools that interface with the CCER Ontology. Section 4.1 discusses version assignment, and Section 4.2 discusses the impact of version changes to ontology users.

4.1 VERSION ASSIGNMENT BY DEVELOPERS

CCER Ontology consists of ontologies developed by CCER team and also developed by third-parties. When an ontology developer wants to make changes to an ontology within CCER, a lot of design issues must be considered. For example, “when is an ontology change considered a *major* upgrade, and hence needs a namespace change?” This section provides a summary of guidelines that answers these kinds of questions. Figure 4 illustrates how CCER ontology developers determine the version number for a new CCER Topic when different kinds of changes or updates are made. Six major use cases along with some variants are diagrammed in Figure 4.

1. *Extending or refining an existing ontology (1).* This use case covers situations when classes or properties in an existing ontology have been added (A), removed (B), or modified (C). Adding ontology classes or properties (A) poses the fewest problems and will always result in a minor version change that is fully backward compatible (J). When one or more ontology classes or properties is removed (B), the result will always be a minor version change that is fully backward compatible so long as deprecation of the classes is properly performed (K). Handling modifications to existing ontology classes or properties (C) is conditional. Depending on an assessment of whether the impact of changes is small or large, this will require a minor (J) or major (K) version change to the CCER topic release. If the version change is a major one (K), changes to class and properties will *not* be fully backward compatible. Deprecated classes, properties, and instances (and the associated deprecation annotations) will need to be removed (L), the major version number in the ontology’s versionInfo annotation will need to be incremented (with minor and pico version numbers set to 0), and a new folder needs to be created for the incremented major version (M). This new folder will need to be created with copies of updated topic files and any dependent ontology files from the previous version (updated as required) — in other words, everything that is still necessary for the new major version. This will require the creation of new namespace URIs for the incremented version ontologies and the modification of imported URIs in dependent ontology files (G). All such dependent ontologies will then need to undergo a major version change (K) and all the subsequent actions that this entails (L, I, M, G);

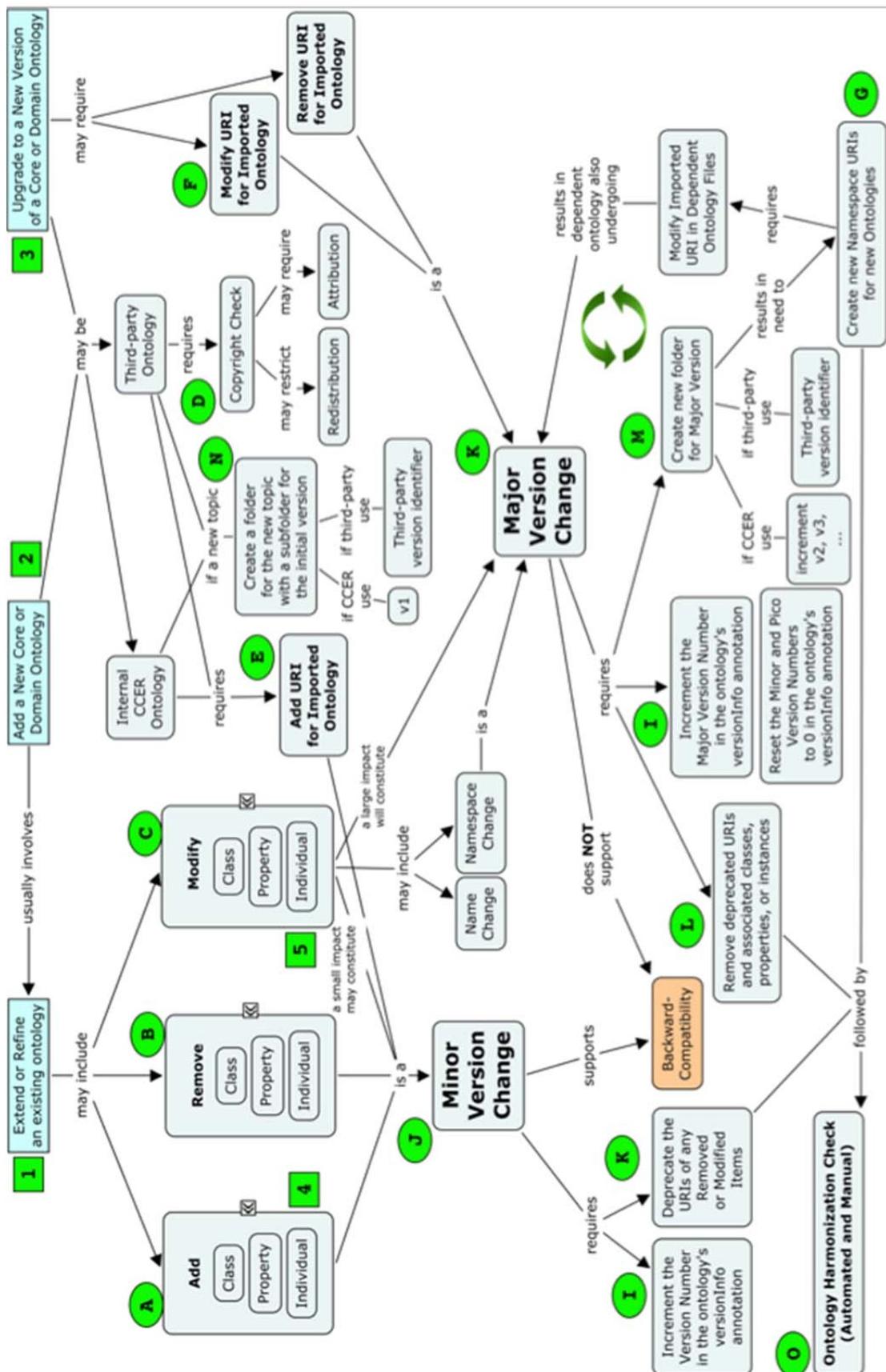


Figure 4: Assigning Version Numbers

2. *Adding a new core or domain ontology (2).* The first thing that should be done, in case this is a third-party ontology, is a check to make sure no copyrights are violated (D). Bridge ontologies may need to be created to use these third party ontologies. This may restrict redistribution and will require attribution to the third party in documentation. Adding the ontology will also require adding an imported ontology URI to existing ontologies that use it, requiring a minor version change in these dependent ontologies (E). This change, however, will be fully backward compatible. If it is a new topic, it will also require creating a new folder with a subfolder for the initial version (N);
3. *Upgrading to a new version of an imported core or domain ontology (3).* This use case covers situations when there is a version upgrade of an imported ontology. Such a change requires a modification of the imported ontology URI (F) and thus necessitates a major version change to the CCER topic (K). Deprecated classes, properties, and instances (and the associated deprecation annotations) will need to be removed, the major version number in the ontology's versionInfo annotation will need to be incremented (with minor and pico version numbers set to 0), and a new folder for the incremented major version (M) need to be created. This new folder will need to be created with copies of: a. updated topic files b. any dependent ontology files from the previous versions (updated as required) — in other words, everything that is still necessary for the new major version. This will require the creation of new namespace URIs for the incremented version ontologies and the modification of imported URIs in dependent ontology files (G). All such dependent ontologies will then need to undergo a major version change (J) and all the subsequent actions that this entails (L, I, M, G);
4. *Adding Named Individuals (4).* These sorts of modifications should require only a change to minor version number for core and domain ontologies, and will almost always be fully backward compatible, though they might impact software tools using the ontology.
5. *Removing, or modifying individuals (5).* Depending on an assessment of whether the impact of changes is small or large, this will require a minor (J) or major (K) version change to the CCER ontology release. If the version change is a major one (J), the new release will not be fully backward compatible. Deprecated classes, properties, and instances (and the associated deprecation annotations) will need to be removed, the major version number in the ontology's versionInfo annotation will need to be incremented (with minor and pico version numbers set to 0), and a new folder for the incremented major version (M). This new folder will need to be created with copies of: a. updated topic files b. any dependent ontology files from the previous version (updated as required) — in other words, everything that is still necessary for the new major version. This will require the creation of new namespace URIs for the incremented version ontologies and the modification of imported URIs in dependent ontology files (F). All such dependent ontologies will then need to undergo a major version change (K) and all the subsequent actions that this entails (L,

I, M, G). Often, when use cases are updated, the corresponding ontology file will be upgraded to a newer major version number.

6. *Bug fixes.* These sorts of modifications should require only a change to the pico version number and should almost always be fully backward compatible.

Both major and minor version updates require a manual or automated harmonization check (O).

4.2 ONTOLOGY EVOLUTION FOR USERS

When a user of CCER Ontology observes changes in the version of a Topic in CCER Ontology, then the question arises as to how the changes in CCER Ontology impacts the user. This section addresses this question.

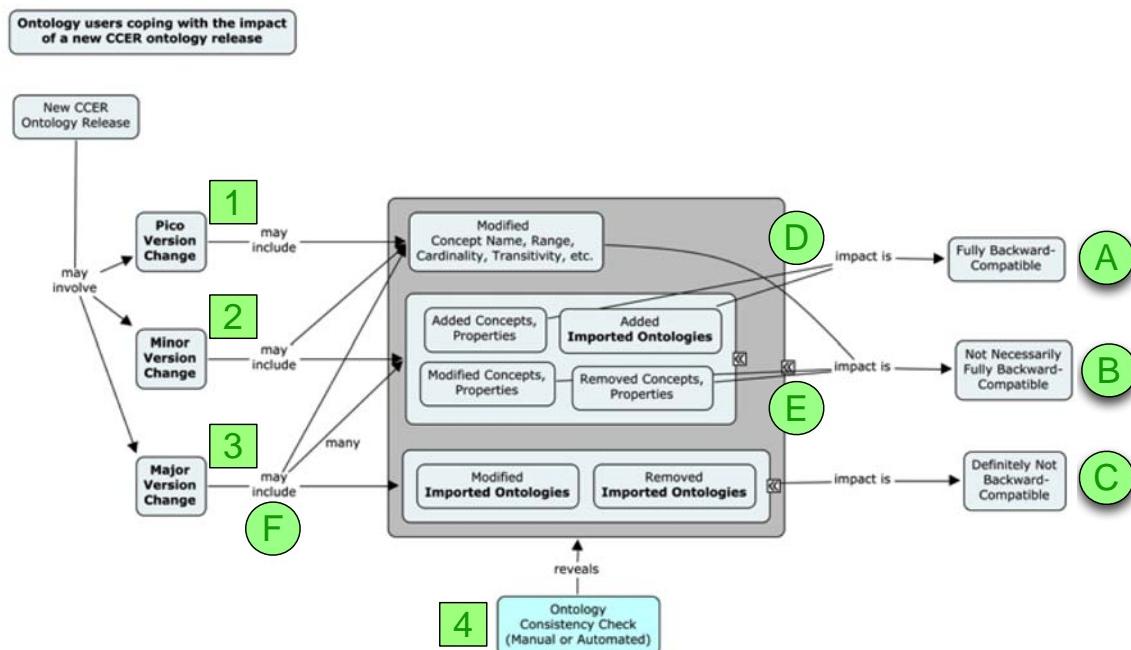


Figure 5: Impact of Version Upgrade on Users

The figure above is designed to help CCER ontology users ascertain the impact of a new release of the CCER Ontology. The users may be concerned that the instance ontology they generated is conformant with the newer version of CCER Ontology, or may be concerned that some changes they intend to make will break the conformance to the version. Only in the case of added classes, properties, and imported ontologies, when there is a minor version change, can guarantees be made about full backward compatibility. Minor version changes with modified or removed classes and properties may result in situations where backward compatibility cannot be guaranteed. However, if the practices above for deprecating classes and properties have been followed, backward compatibility can be maintained. Backward compatibility is definitely not guaranteed when imported ontologies have been modified or removed and a major version change to the bundle is indicated.

1. *Pico version change.* Pico version changes may include changes to class names, ranges, cardinality, transitivity, etc. They may also involve individuals that are added, modified, or removed. In almost all cases, such changes will be fully backward compatible, but this is not guaranteed to be the case (B). For this reason, manual and/or automated consistency checks should always be performed (4);
2. *Minor version change.* Minor version changes will be fully-backward compatible (A) in the case of updates that consist only of some combination of the following: 1) added classes to existing ontologies and/or added imported ontologies (D); and 2) removed classes that have been deprecated in a manner consistent with practices outlined in this document. In the case of modified or removed classes (E), the changes will not necessarily be fully backward compatible (B). For this reason, manual and/or automated consistency checks should always be performed (4).
3. *Major version change.* Major version changes may include modified or removed ontologies, in addition to any of the kinds of changes discussed in 1 or 2 above. No guarantees about backward compatibility can be made in such cases (C) and manual and/or automated consistency checks should always be performed (4).

Ontology consistency check. Manual or automated checks may reveal any of the types of changes described above and need to be handled accordingly.

5. CONCLUSION

Versioning ontologies is a complex task, yet is necessary to maintain the integrity of ontologies, and applications that depend on them. In this document, we described the guidelines CCER team adopted for versioning of CCER Ontology. The guidelines aim to reduce the complexity of versioning by stipulating correlation of folder structure, and namespace URIs, and a defined process for version upgrades. We also describe guidelines on how the users of an ontology may assess an impact of a version upgrade. We introduce the idea of a Topic as the unit for versioning in contrast to versioning the CCER Ontology. Understanding the state of the versions of the Topics in an ontology can be time consuming and tedious, and automated tools will be needed to effectively manage versioning. Future work will include creating tools for instance configuration comparison, topic evolution analysis, process for Topic submissions, and automation for ontology merging.

BIBLIOGRAPHY

- [CSTL1] Peter Haglich, Robert Grimshaw, Steven Wilder, Marian Nodine, and Bryan Lyles. Cyber scientific test language. In ISWC 2011, volume 7032 of Lecture Notes in Computer Science, pp 97–111. Springer Berlin Heidelberg, 2011.
- [CCER1] Damodaran, Suresh K., and David Tidmarsh. Model Based Verification of Cyber Range Event Environments. In *Spring Simulation Multi-Conference*, Pasadena, CA, USA, April 2016 (to be presented).
- [CRIS1] Suresh K. Damodaran and Jerry M. Couretas. Cyber Modeling & Simulation for Cyber-Range Events. In *Summer Simulation Multi-Conference*, Chicago, IL, USA, 2015.
- [NOY1] Noy, Natalya Fridman, and Tania Tudorache. Collaborative Ontology Development on the (Semantic) Web. *AAAI Spring Symposium: Symbiotic Relationships between Semantic Web and Knowledge Engineering*. 2008.
- [LHK09] Matthias Loskyll, Dominikus Heckmann, and Ichiro Kobayashi. "UbisEditor 3.0: Collaborative Ontology Development on the Web." *Proceedings of the Hypertext 2009 workshop on Web*. 2009.
- [SEA1] W3C, Search Engines, https://www.w3.org/wiki/Search_engines, Accessed February 02, 2016
- [VURI]Community:Versioning and URIs
http://ontologydesignpatterns.org/wiki/Community:Versioning_and_URIs. Accessed February 02, 2016
- [RGZH] Jiménez-Ruiz, E., Cuenca Grau, B., Zhou, Y., Horrocks, I.: Large-scale interactive ontology matching: Algorithms and implementation. In Proc. of ECAI (2012)
- [GJKZ] Grau, B.C., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D.: Ontology evolution under semantic constraints. In: Proc. of International Conference on Principles of Knowledge Representation and Reasoning (KR), Rome, Italy (2012)
- [RGHB] E. Jiménez Ruiz, B. Cuenca Grau, I. Horrocks, R. Berlanga, Supporting concurrent ontology development: Framework, algorithms and tool, Data & Knowledge Engineering, Volume 70, Issue 1, January 2011, Pages 146-164, ISSN 0169-023X, <http://dx.doi.org/10.1016/j.datak.2010.10.001>.
- [OWL2] W3C, OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition), W3C Recommendation 11 December 2012. <https://www.w3.org/TR/owl2-syntax/> Accessed, February 02, 2016

[OWL3] W3C, *OWL Web Ontology Language Use Cases and Requirements*, W3C Recommendation 10 February 2004, <https://www.w3.org/TR/2004/REC-webont-req-20040210/>

[OVR1] W3C, Ontology Versions, https://www.w3.org/2007/OWL/wiki/Ontology_Versions, Accessed, February 02, 2016

[RDF1] Resource Description Framework (RDF) <https://www.w3.org/RDF/>

[UBJ2004] Uszok, Andrzej, Jeffrey M. Bradshaw, and Renia Jeffers. Kaos: A policy and domain services framework for grid computing and semantic web services. *Trust Management*. Springer Berlin Heidelberg, 2004. 16-26.

[TIME] W3C, Time Ontology in OWL. W3C Working Draft 27 September 2006
<https://www.w3.org/TR/owl-time/>

[NML] Jeroen van der Ham, Freek Dijkstra, Roman Łapacz, Jason Zurawski. “Network Markup Language Base Schema version 1.” Open Grid Forum, GFD-R-P.206, Grid Final Draft, May 2013.